Cache aware multigrid for variable coefficient elliptic problems on adaptive mesh refinement hierarchies

C. C. Douglas[1], J. Hu[2], J. Ray[3], D. T. Thorne[4], R. S. Tuminaro[5]

[1] *University of Kentucky, Department of Computer Science, 325 McVey Hall, Lexington, KY 40506-0045, USA; also, Yale University, Department of Computer Science, P.O. Box 208285, New Haven, CT 06520-8285, USA, douglas-craig@cs.yale.edu.*
[2] *Sandia National Laboratory, Mail Stop 9217, Livermore, CA 94550, USA, jhu@ca.sandia.gov.*
[3] *Sandia National Laboratory, Combustion Research Facility, Mail Stop 9056, P.O. Box 969, Livermore, CA 94550, USA, jairay@ca.sandia.gov.*
[4] *University of Kentucky, Department of Computer Science, 325 McVey Hall, Lexington, KY 40506-0045, USA, thorne@ccs.uky.edu.*
[5] *Sandia National Laboratory, Mail Stop 9217, Livermore, CA 94550, USA, tuminaro@ca.sandia.gov.*

SUMMARY

We derive a multilevel algorithm to solve variable coefficient elliptic boundary value problems on adaptively refined structured meshes, and we design a cache optimized version of this algorithm. The operations are optimized to exploit the cache memory subsystem. We present numerical results demonstrating the efficiency of the cache optimization.

KEY WORDS: cache aware, multigrid, elliptic, adaptive mesh refinement, hierarchies

1. Introduction

Patch-based adaptive mesh refinement [2, 3] has become an attractive technique for spatial discretization in a host of scientific simulations, ranging from shock hydrodynamics [12] to combustion [5] and plasma physics [15]. The ability to place a refined patch (with a Cartesian mesh) at any arbitrary location in the domain obviates the need for mesh stretching and promotes its use with higher-order stencils [13]. Conformal transformation techniques have extended this technique to nonrectangular/cuboid domains of sufficient complexity to meet the needs of scientific (as opposed to engineering) simulations [8].

Contract/grant sponsor: Sandia National Laboratories

Contract/grant sponsor: National Science Foundation

Contract/grant sponsor: Intel

Contract/grant sponsor: Hewlett-Packard Corporations

A large number of scientific problems (e.g. combustion, incompressible hydrodynamics, magnetohydrodynamics etc) frequently pose a Poisson problem with variable coefficients as a part of their solution strategy. While this could be formulated as a giant $\mathbf{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ problem and treated with iterative methods (e.g. Krylov methods), the multilevel nature of the grid strongly suggests multigrid approaches. Further, the (relatively) small data set associated with a patch lends this approach to cache-based optimization of numerical operations local to a patch i.e. smoothing. In this paper, we use a combination of adaptive refinement [2, 3, 14] and multilevel [4, 9, 10] procedures to solve variable coefficient elliptic boundary value problems of the form

$$\begin{cases} \mathcal{L}(\phi) = \rho \text{ in } \Omega, \\ \mathcal{B}(\phi) = \gamma \text{ on } \partial\Omega, \end{cases} \tag{1}$$

subject to standard conditions that ensure ellipticity and well posedness [1]. The solution procedure is derived from the adaptive mesh refinement process, not from the multigrid procedure. Hence, we use notation that is common in the adaptive mesh refinement (AMR) community.

This research focuses on

1. Implementing cache aware optimizations to multigrid in an AMR context,
2. Modifying a well known AMR multigrid algorithm [9] to do only post-smoothing so that the cache aware optimizations will have a greater effect.

Cache aware algorithms are designed to minimize the number of times data goes through cache, thereby increasing the efficiency of the algorithm. Cache memories are much faster than main memory, so the CPU can be kept more busy when it is getting data from cache memories. In an AMR context, we modify Gauss-Seidel so that all the data required by the smoothing iterations needs to be brought into a cache only once, not many times, and it still gets the exact same answer as the standard algorithm (i.e. bitwise compatibility). In order to further improve the efficiency of cache usage, we do only post-smoothing and combine the residual computation with the smoother. In this way all of the smoothing *and* the residual computation for each level of a V cycle can be accomplished while bringing the data into cache just once.

Doing post-smoothing only is a substantial change over the AMR algorithm used in [9]. It is especially useful when implementing cache aware algorithms as discussed in §5. We see far better cache effects when more smoothing iterations are done consecutively. Further, reducing the work that is done outside of the smoother means that speeding up the smoother will have a greater impact on the whole algorithm.

The variable coefficient problem that we present here is different from a constant coefficient or Poisson problem since there is the coefficient matrix that has to be fetched into cache along with the solution and right hand side. This poses the problem of how to store the coeffiencent matrix to facilitate the cache aware optimizations as discussed in §5.

This paper assumes the reader is familiar with discretization and numerical solution of partial differential equations [6, 11, 17].

In §2, we provide a mathematical background. In §3, we present a 2D example. In §4, the multilevel adaptive mesh refinement method is described. In §5, we discuss cache aware Gauss-Seidel and the V cycle. In §6, we present numerical results.

Many very highly technical intermediate details have been left out of this article. The omitted material is in [16].

2. Mathematical Background

The basic algorithms are geometrically inspired. Definitions are based on a domain (or subdomain) rather than a grid perspective. This is standard in adaptive grid refinement literature but less standard in multigrid literature.

We begin by assuming that Ω is overlaid by a union of tensor product meshes $\Lambda^{1,j}$, $j = 1, \ldots, n_1$ that form a grid in Ω:

$$\Lambda^1 = \bigcup_{j=1}^{n_1} \Lambda^{1,j}, \quad \text{where } \Lambda^1 \subset \Omega.$$

Normally $n_1 = 1$. However, the method works for $n_1 > 1$, too. This is referred to as the level 1, or coarsest grid. Operators are defined on it later.

An adaptive mesh refinement procedure is used to define many *patches*. The set of local grid patches corresponding to $\ell - 1$ refinements ($1 < \ell \leq \ell_{max}$) is denoted

$$\Lambda^\ell = \bigcup_{j=1}^{n_\ell} \Lambda^{\ell,j} \quad \text{and} \quad \Lambda^{\ell_{max}+1} = \emptyset.$$

The $\Lambda^{\ell,j}$ are tensor product meshes that have been obtained by adaptively refining the $\Lambda^{\ell-1,j}$ meshes. The definition for $\Lambda^{\ell_{max}+1}$ is a convenience that simplifies a number of the algorithms we define throughout this paper. We define the domains Ω^ℓ and $\Omega^{\ell,j}$ as the minimum domains that include Λ^ℓ and $\Lambda^{\ell,j}$, respectively. Normally, Ω^ℓ is a union of disconnected subdomains (one subdomain corresponding to each level ℓ patch). Note that within an adaptive grid refinement code ℓ_{max} can change (increase or decrease) during the course of solving an actual problem.

The AMR procedure defines a composite grid, ${\Lambda_c}^{\ell_{max}}$, and more generally, a composite grid hierarchy, $1 < \ell \leq \ell_{max}$, by

$${\Lambda_c}^\ell = \bigcup_{i=1}^{\ell} (\Lambda^i - \mathcal{P}(\Lambda^{i+1})), \tag{2}$$

where $\mathcal{P}$ is the projection operator discussed below. The ℓ^{th} composite grid, ${\Lambda_c}^\ell$, contains all points from the ℓ^{th} level patches, Λ^ℓ, as well as additional points from regions not covered by the patches. The new grid points correspond to mesh points from patches on lower levels, always taking from patches on the closest possible level.

We use projection and refinement operators $\mathcal{P}$ and $\mathcal{R}$, respectively. The notation is standard adaptive mesh refinement notation but is different from multigrid notation (where the symbols are unfortunately reversed). The operators are used interchangeably with either domains Ω^ℓ or grids Λ^ℓ. In terms of domain and grid superscripts, $\mathcal{P}$ *projects* from "fine to coarse," i.e., $\ell \to \ell - 1$ and $\mathcal{R}$ *refines* from "coarse to fine," i.e., $\ell \to \ell + 1$.

We require nesting of domains

$$\Omega^{\ell_{max}} \subseteq \Omega^{\ell_{max}-1} \subseteq \cdots \cdots \Omega^1 \equiv \Omega,$$

which can be written as

$$\mathcal{R}(\mathcal{P}(\Omega^{\ell+1})) \subseteq \Omega^{\ell+1} \quad \text{and} \quad \mathcal{P}(\Omega^{\ell+1}) \subseteq \Omega^\ell$$

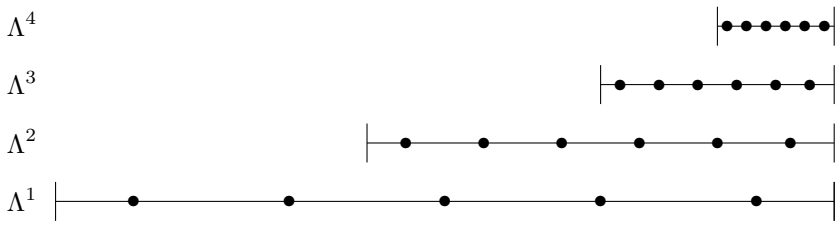


Figure 1. A sample grid hierarchy.

and

$$\Omega = \bigcup_{\ell=1}^{\ell_{max}} (\Omega^\ell - \mathcal{P}(\Omega^{\ell+1})), \quad \text{where } \Omega^{\ell_{max}+1} = \emptyset.$$

The use of tensor product meshes allows for fairly straightforward finite difference and finite volume stencils to define the discrete operator. At internal patch boundaries, however, some care must be taken. We define *ghost points* near internal patch boundaries and use quadratic interpolation to acquire values at these points so that locally equispaced unknowns are available for use with regular stencils within most of the computations. When computing composite grid residuals, however, more complicated stencils are needed for coarse points adjacent to finer grid patches. This is formally covered in §3. The main idea is to use the same interpolated values for the stencils on both the fine grid side and the coarse grid side when updating points that are adjacent to a coarse-fine interface. Hence, the fluxes used to approximate the operator across the coarse-fine interfaces are matched and continuity of the first derivate (i.e. C^1 continuity) is enforced. This is referred to as *flux matching*. The C^1 continuity at the boundary between the coarse and fine subdomains can be precisely defined in terms of the derivatives of the quadratic functions that interpolate the ghost points. See §3 for details of the ghost point interpolation procedure. The key point is that enforcing C^1 continuity preserves the second order convergence of the method. Especially for problems with severe fronts or near discontinuities, C^0 continuity alone is not always sufficient. The boundaries of the domains, $\{\partial\Omega^\ell\}$, are required to meet the condition

$$\partial\Omega^{\ell+1} \cap \partial\Omega^\ell \subseteq \partial\Omega$$

which ensures that the flux matching procedure that we use is well defined and of the right approximation order near patch boundaries in the interior of Ω [9].

We approximate the solution to (1) using a multigrid algorithm to numerically solve the finite dimensional problem

$$\mathcal{L}_c^{\ell_{max}} \phi_c^{\ell_{max}} = \rho_c^{\ell_{max}}, \tag{3}$$

where $\mathcal{L}_c^{\ell_{max}}$ is a matrix representing the discretization on $\Lambda_c^{\ell_{max}}$, $\phi_c^{\ell_{max}}$ are the unknowns, and $\rho_c^{\ell_{max}}$ is the right hand side. Conceptually, the grid hierarchy used within the multigrid procedure is the composite grid hierarchy defined above. In practice, we design the implementation to be patch based (see §4). A simple 1D AMR patch hierarchy is illustrated in Figure 1. In the figure, the dots represent grid points and the vertical lines represent either a physical boundary or a patch boundary. A patch is a maximal set of contiguous grid points on a given level of the grid hierarchy. There can be multiple patches per level, although the

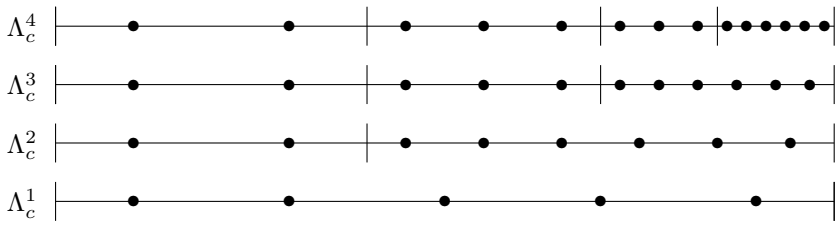


Figure 2. Composite grid hierarchy corresponding to the sample grid in Figure 1.

grid hierarchy in Fig. 1 has only one patch per level. Figure 2 illustrates the corresponding composite grid hierarchy for the simple 1D example. Each level in this composite grid hierarchy is a composite grid defined in (2). Equation (3) is defined on the highest level composite grid, namely $\Lambda_c^{\ell_{max}}$. Operators $\mathcal{L}_c^\ell$ are matrices representing the discretizations on composite grids Λ_c^ℓ for all ℓ. In §4, we define patch based versions of the discrete operator.

3. An Example

This section describes the tools that are needed to define patch based versions of the discrete operators used by the multilevel method presented in §4. The specific form of equation (1) that this discussion will address is

$$\begin{cases} -\nabla \cdot (a \nabla u) = \rho, & \text{in } \Omega, \\ u = \gamma, & \text{on } \partial\Omega, \end{cases}$$

where Ω is a rectangular domain and γ is a constant. In addition, this example will assume isotropic mesh spacing $h = \Delta x = \Delta y$. To discretize the variable coefficient problem we use a control volume approach [11]. Fig. 3 illustrates the stencils for an interior cell, an edge cell and a corner cell. The region within the dashed lines is the control volume, denoted by V. The

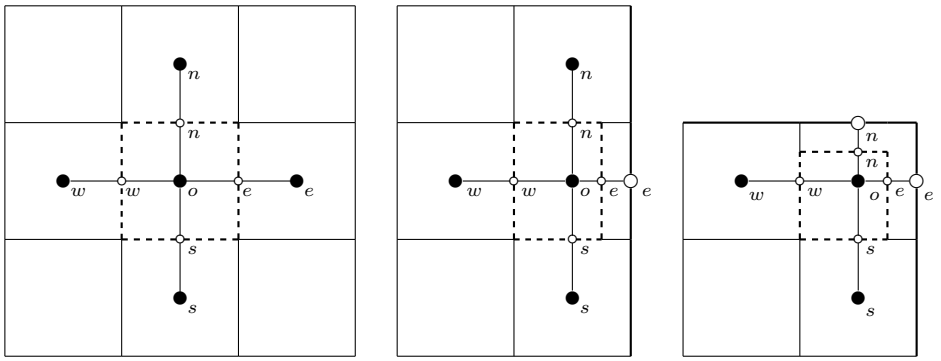


Figure 3. Stencils. The dark lines in (b) and (c) represent boundaries.

boundary of the control volume, represented by the dashed lines, is denoted by ∂V. Formulas

for the stencils are achieved by discretizing the integral form

$$-\int_{\partial V} a\frac{\partial u}{\partial n} = \int_V \rho.$$

The formula for interior cells is the usual, the same as for vertex centered grids. The discretization for the edge stencil gives

$$\begin{aligned}\frac{a_n(u_n-u_o)}{\Delta y}\cdot\frac{3}{4}\Delta x \quad &- \quad \frac{a_s(u_o-u_s)}{\Delta y}\cdot\frac{3}{4}\Delta x + \\ \frac{a_e(u_e-u_o)}{\frac{1}{2}\Delta x}\cdot\Delta y \quad &- \quad \frac{a_w(u_o-u_w)}{\Delta x}\cdot\Delta y = -\frac{3}{4}\Delta x\Delta y\rho_o,\end{aligned}$$

which reduces to

$$(a_n+a_s+\frac{8}{3}a_e+\frac{4}{3}a_w)u_o - a_nu_n - a_su_s - \frac{4}{3}a_wu_w - \frac{8}{3}a_eu_e = h^2\rho_o,$$

assuming $h=\Delta x=\Delta y$. The discretization for the corner stencil gives

$$\begin{aligned}\frac{a_n(u_n-u_o)}{\frac{1}{2}\Delta y}\cdot\frac{3}{4}\Delta x \quad &- \quad \frac{a_s(u_o-u_s)}{\Delta y}\cdot\frac{3}{4}\Delta x + \\ \frac{a_e(u_e-u_o)}{\frac{1}{2}\Delta x}\cdot\frac{3}{4}\Delta y \quad &- \quad \frac{a_w(u_o-u_w)}{\Delta x}\cdot\frac{3}{4}\Delta y = -\frac{9}{16}\Delta x\Delta y\rho_o,\end{aligned}$$

which reduces to

$$(\frac{8}{3}a_n+\frac{4}{3}a_s+\frac{8}{3}a_e+\frac{4}{3}a_w)u_o - \frac{8}{3}a_nu_n - \frac{4}{3}a_su_s - \frac{4}{3}a_wu_w - \frac{8}{3}a_eu_e = h^2\rho_o,$$

when $h=\Delta x=\Delta y$.

In order to apply the discrete operator at interfaces between coarse and fine grids, *ghost points* are interpolated around patches. These ghost points are used to complete the stencils on the fine grid side of interfaces. A *flux matching* procedure, employing the ghost points, is used to define the operator on the coarse grid side of an interface. The ghost point interpolation and flux matching procedures are described in the following subsections.

3.1. Ghost Point Interpolation

In order to apply regular stencils at the boundary points of patches and enforce flux matching at the coarse-fine interfaces, we need to interpolate values at ghost points around the edges of the patches. This section explains how to do these interpolations. They are necessary for both of the patch based operators $\mathcal{L}^\ell$ and $\mathcal{L}^{nf,\ell}$ defined in §4.

3.1.1. One Coarse-Fine Interface Values are needed at the $\otimes$ ghost points for the one-sided coarse-fine interface illustrated in Figure 4(a). There are two steps.

Step 1: Compute values for the the $\oslash$ points. Let $a=u(\bullet_A)$, $b=u(\bullet_B)$ and $c=u(\bullet_C)$, and let h be the mesh spacing on the finer grid. We derive a C^1 quadratic interpolation formula

$$u(x) = c_0 + c_1x + c_2x^2,$$

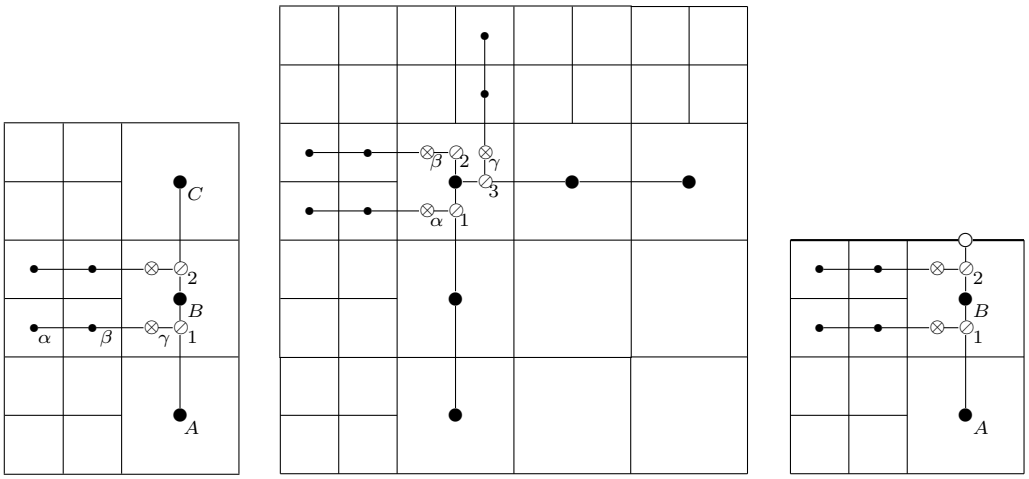


Figure 4. Interpolation of Ghost Points. The dark edge in (c) indicates a boundary.

where $c_0 = a$, $c_1 = \frac{1}{h}(b - \frac{1}{4}c - \frac{3}{4}a)$, and $c_2 = \frac{1}{h^2}(\frac{1}{8}c - \frac{1}{4}b + \frac{1}{8}a)$. Then

$$u(\oslash_1) = c_0 + c_1(\frac{3}{2}h) + c_2(\frac{3}{2}h)^2 \text{ , and } u(\oslash_2) = c_0 + c_1(\frac{5}{2}h) + c_2(\frac{5}{2}h)^2.$$

Step 2: Compute values for the $\otimes$ points. Let $a = u(\bullet_\alpha)$, $b = u(\bullet_\beta)$ and $c = u(\oslash_1)$, and let h be the mesh spacing on the finer grid. Again, we derive a C^1 quadratic interpolation formula using

$$u(x) = c_0 + c_1 x + c_2 x^2,$$

where $c_0 = a$, $c_1 = -\frac{1}{15h}(21a - 25b + 4c)$, and $c_2 = \frac{2}{15h^2}(3a - 5b + 2c)$. Then

$$u(\otimes_\gamma) = c_0 + c_1(2h) + c_2(2h)^2.$$

The procedure is analogous for the other $\otimes$ point.

3.1.2. Two Coarse-Fine Interfaces Ghost point interpolation at a two-sided coarse-fine interface is illustrated in Figure 4(b). Again, values are needed at the $\otimes$ points, and there are two steps. The notation $c_{\{0,1,2\}}$ refers generically to the coefficients associated with each interpolation. The values of the coefficients are *not* the same for every interpolation.

Step 1: Use the same system as in *Step 1* of §3.1.1, once in each direction, to get the coefficents $c_{\{0,1,2\}}$ needed to compute values for the $\oslash$ points.

Step 2: Use the same system as in *Step 2* of §3.1.1, twice in the x-direction and once in the y-direction, to get the coefficents $c_{\{0,1,2\}}$ needed to compute values for the $\otimes$ points.

3.1.3. At a Boundary Ghost point interpolation at a coarse-fine interface adjacent to a boundary is illustrated in Figure 4(c). Once again, values are needed at the $\otimes$ points, and there are two steps.

Step 1: Compute values for the $\oslash$ points as in *Step 1* of the previous sections. Use $a = u(\bullet_A)$, $b = u(\bullet_B)$ and $c = u(\bigcirc)$.

Step 2: Compute values for the $\otimes$ points by the same system as in *Step 2* of §3.1.1.

3.2. Flux Matching

This section introduces the flux-matching computations. Flux matching is used to avoid one-sided derivatives and preserve C^1 continuity. It is necessary for the patch based operator $\mathcal{L}^\ell$ used in §4.

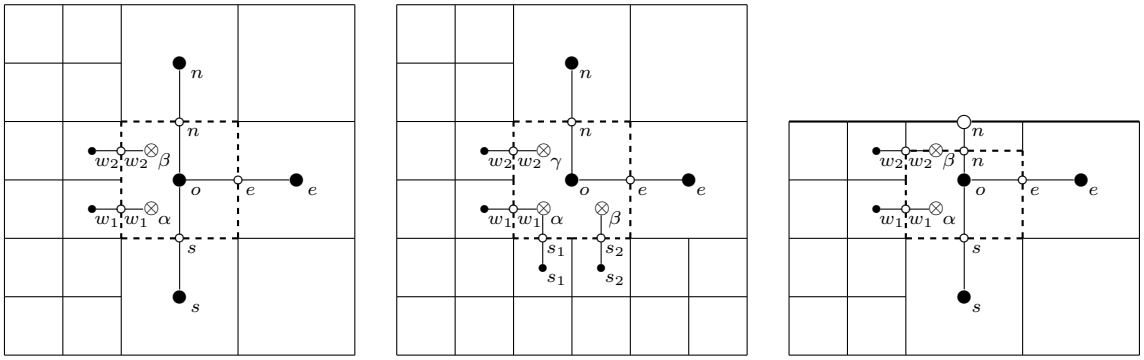


Figure 5. Flux matching for the variable coefficient case. The dark edge in (c) represents a boundary.

See Fig. 5 for illustrations. The regions within the dashed lines are the control volumes, denoted by V. The boundary of the control volumes, represented by the dashed lines, are denoted by ∂V.

The grid points on edges represent coefficient values. The grid points in cells represent solution values, as usual. For example, $a_n = a(\circ_n)$ and $u_n = u(\bullet_n)$.

Discretizing the integral form

$$-\int_{\partial V} a \frac{\partial u}{\partial n} = \int_V \rho$$

gives

$$-(f_n - f_s + f_e - f_w) = h^2 \rho_o,$$

where the fluxes $f_{\{n,s,e,w\}}$ for the different cases are given in the following sections (assuming isotropic mesh spacing $h = \Delta x = \Delta y$).

3.2.1. One Coarse-Fine Interface See Fig. 5(a). The fluxes for this case are

$$\begin{aligned} f_n &= a_n(u_n - u_o), \quad & f_s &= a_s(u_o - u_s), \\ f_e &= a_e(u_e - u_o), \quad & f_w &= a_{w_1}(u_\alpha - u_{w_1}) + a_{w_2}(u_\beta - u_{w_2}). \end{aligned}$$

3.2.2. Two Coarse-Fine Interface See Fig. 5(b). The fluxes here differ from §3.2.1 only in the south flux:

$$f_s = a_{s_1}(u_\alpha - u_{s_1}) + a_{s_2}(u_\beta - u_{s_2}).$$

3.2.3. At a Boundary See Fig. 5(c). The fluxes for this case are

$$\begin{aligned} f_n &= 2a_n(u_n - u_o), \quad & f_s &= a_s(u_o - u_s), \\ f_e &= \tfrac{3}{4}a_e(u_e - u_o), \quad & f_w &= a_{w_1}(u_\alpha - u_{w_1}) + \tfrac{1}{2}a_{w_2}(u_\beta - u_{w_2}). \end{aligned}$$

4. Adaptive Mesh Refinement Multilevel Method

Algorithm 1 AMR Multigrid V cycle.

MG($\ell, e^\ell, r^\ell, \rho^\ell, \phi^{\ell_{max}}$ **)**

if $\ell == 1$ **then**
  $e^\ell \leftarrow S^\ell(e^\ell, r^\ell)$ on Λ^ℓ
  $\phi^{\ell_{max}} \leftarrow \phi^{\ell_{max}} + e^\ell$ on $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$
  return
end if
if $\ell == \ell_{max}$ **then**
  $r^\ell = \rho^\ell - \mathcal{L}^{nf,\ell}(\phi^\ell, \phi^{\ell-1})$
end if
$e^\ell \leftarrow 0$; $e^{\ell-1} \leftarrow 0$
$e^\ell \leftarrow S^\ell(e^\ell, r^\ell)$ on Λ^ℓ
$\phi^{\ell,temp} \leftarrow e^\ell + \phi^{\ell_{max}}$ on $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$
$\hat{r}^\ell \leftarrow r^\ell - \mathcal{L}^{nf,\ell}(e^\ell, e^{\ell-1})$ on Λ^ℓ
$r^{\ell-1} \leftarrow \mathcal{P}^\ell \hat{r}^\ell$ on $\mathcal{P}(\Lambda^\ell)$
$r^{\ell-1} \leftarrow \rho^{\ell-1} - \mathcal{L}^{\ell-1}(\phi^{\ell,temp}, \phi^{\ell-1}, \phi^{\ell-2})$ on $\Lambda^{\ell-1} - \mathcal{P}(\Lambda^\ell)$
MG($\ell - 1, e^{\ell-1}, r^{\ell-1}, \rho^{\ell-1}, \phi^{\ell_{max}}$)
$e^\ell \leftarrow e^\ell + \mathcal{R}^{\ell-1} e^{\ell-1}$
$r^\ell \leftarrow r^\ell - \mathcal{L}^{nf,\ell}(e^\ell, e^{\ell-1})$ on Λ^ℓ
$\bar{e}^\ell \leftarrow 0$
$\bar{e}^\ell \leftarrow S^\ell(\bar{e}^\ell, r^\ell)$ on Λ^ℓ
$e^\ell \leftarrow e^\ell + \bar{e}^\ell$ on Λ^ℓ
$\phi^{\ell_{max}} \leftarrow \phi^{\ell_{max}} + e^\ell$ on $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$

The AMR multigrid algorithm is shown in Alg. 1. The patch based discrete operators $\mathcal{L}^\ell$ and $\mathcal{L}^{nf,\ell}$ are restrictions of the composite grid operator onto the patches on level ℓ with the interfaces between coarse and fine patches handled by ghost points and flux matching as shown in §3. The operator $\mathcal{L}^\ell$ operates on $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$ using ghost points to complete the stencil at the exterior boundary of the patch and using flux matching to update the points at interior boundaries where there are finer grid patches. The operator $\mathcal{L}^{nf,\ell}$ operates on entire patches Λ^ℓ ignoring finer levels and, hence, does not require the flux matching procedure.

4.1. Post-smoothing only

Alg. 1 can be sped up considerably by only doing post-smoothing, as illustrated in Alg. 2. There are several advantages to this approach:

1. The initial guess only needs to be set (to 0) on the base grid.

Algorithm 2 Multigrid V cycle with post-smoothing only

MG $(\ell, e^\ell, r^\ell, \rho^\ell, \phi^{\ell_{max}}\ )$

if $\ell == \ell_{max}$ **then**
  $r^\ell = \rho^\ell - \mathcal{L}^{nf,\ell}(\phi^\ell, \phi^{\ell-1})$
end if
if $\ell == 1$ **then**
  $e^1 \leftarrow 0$
  Solve $\mathcal{L}_c^1 e^1 = r^1$ on Λ^1
else
  $r^{\ell-1} \leftarrow \mathcal{P}^\ell r^\ell$ on $\mathcal{P}(\Lambda^\ell)$
  $r^{\ell-1} \leftarrow \rho^{\ell-1} - \mathcal{L}^{\ell-1}(\phi^\ell, \phi^{\ell-1}, \phi^{\ell-2})$ on $\Lambda^{\ell-1} - \mathcal{P}(\Lambda^\ell)$
  MG $(\ell-1, e^{\ell-1}, r^{\ell-1}, \rho^{\ell-1}, \phi^{\ell_{max}}\ )$
  $e^\ell \leftarrow \mathcal{R}^\ell e^{\ell-1}$ {Includes interpolation of ghost points.}
  $e^\ell \leftarrow S^\ell(e^\ell, r^\ell)$ on Λ^ℓ
end if
$\phi^{\ell_{max}} \leftarrow \phi^{\ell_{max}} + e^\ell$ on $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$

2. There is no need for a temporary correction $\phi^{\ell,temp}$ on the projection side of the V cycle.
3. $\hat{r}$ does not need to be computed.
4. The residual is only computed on one side of the V cycle.
5. There is no correction after interpolation and no need for an intermediate correction step before updating $\phi^{\ell_{max}}$.
6. The residual is not updated on the interpolation side of the V cycle.

5. Cache Aware Gauss-Seidel and V Cycle

Consider naturally ordered Gauss-Seidel restricted to matrices A_j which are based on discretization methods which are local to only 3 neighboring rows of the grid. Partition the grid into blocks of ℓ rows, and let m be the number of smoothing iterations required. It is necessary that $\ell + m - 1$ rows of an $N \times N$ grid G fit entirely into cache simultaneously and that $m < \ell$.

There are two special cases to the cache aware algorithm: the first block of rows and the rest of the blocks.

The first case is for the first ℓ rows of the grid. The data associated with rows 1 to ℓ is brought into cache. The data in rows 1 to $\ell - m + 1$ are updated m times, and the data in rows j, $\ell - m + 2 \leq j \leq \ell$, are updated $\ell - j + 1$ times.

The second case is for the rest of the blocks of ℓ rows of the grid. Once the first block of grid rows is partially updated, we have a second block to update and must also finish updating the first block of grid rows. After the i^{th} update in the second block, we can go back and update

rows ℓ down to $\ell - i + 1$ in the first block of rows, always performing the updates in the order that preserves the dependencies in the standard iteration (so that the cache aware iteration will achieve bitwise the same answer as the standard iteration). This procedure is repeated in the remaining blocks of rows until all the blocks are updated. In effect, this is a domain decomposition methodology applied to the standard iteration. The result is that m updates are done while bringing all the data through cache only one time.

5.1. Multigrid with the residual computation integrated into the smoother

Integrating the residual computation with the cache-aware smoother can give better cache-effects in multigrid. Call this the *combined smoother*. The combined smoother is implemented for the post-smoothing only version of the algorithm.

The combined smoother computes the residual to be used in the next V cycle. It can only compute the part of the residual that is not covered by finer patches. On the projection side of the V cycle, the projection of the part of the residual that is from finer patches must now include the application of the flux matching procedure, because information needed for the flux matching is not available when the combined smoother is called.

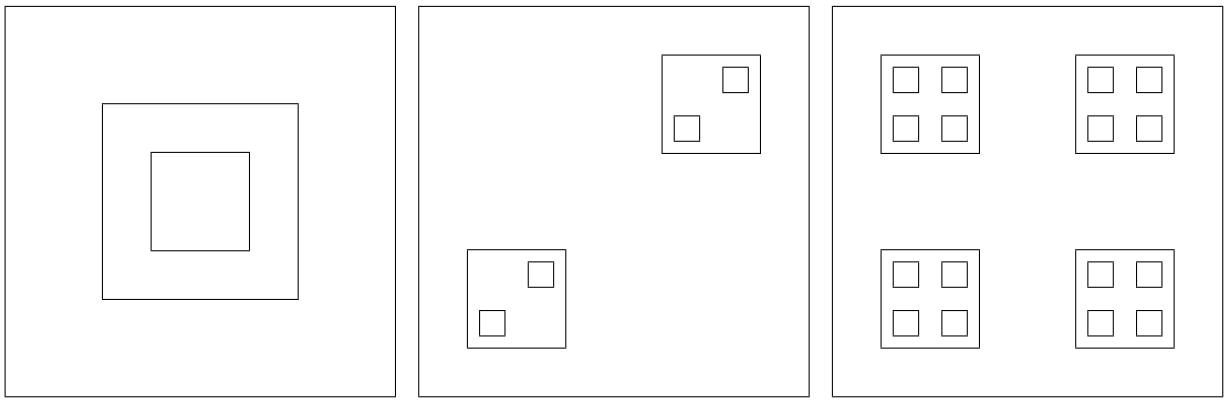
A complication to interleaving the residual computation with the cache aware smoother is that, in the V cycle, the residual is updated *and* concurrently used as the right hand side for the Gauss-Seidel updates. So the residual updates need to be postponed long enough to avoid changing the right hand side for a subsequent Gauss-Seidel update. In addition, residual updates at the patch boundaries require updated ghost points around the patch, so ghost point interpolation must also be interleaved with the cache aware smoother.

5.2. Effects of the Coefficient Matrix

The coefficient matrix has an effect on the cache optimizations since it has to go through cache along with the solution and right hand side. In an attempt to minimize the effect, we tried storing the right hand side in the coefficient matrix in an interleaved manner. Hence, for a given grid point, the coefficients and the right hand side needed to update that point are contiguous in memory. However, the split residual computation complicates this approach. On every level, there are parts of the domain where the residual has to be computed from the right hand side, *and* there are parts of the domain where the residual is projected from finer levels. Depending on the context, the right hand side entries in the coefficient matrix might need to be either the original right hand side or the current residual. Updating the state for a given context involves copies which slow the algorithm. Results in the AMR context are better when the right hand side and residual are separate from the coefficient matrix.

6. Numerical Results

This section shows results on the hierarchies illustrated in Fig. 6 and also on a full domain refinement hierarchy (i.e. regular, non-AMR multigrid). The refinement patterns are not meaningful to the nature of the problem being solved here. They are contrived merely to demonstrate the behavior of the algorithm. The AMR base grid in 6(a) is 128×512 grid points, and there are three levels of refinement above that (although only two are illustrated). The AMR base grid in 6(b) and 6(c) is 256×1024 grid points, and there are four levels of



(a) (b) (c)

Figure 6. (a) One refinement per patch. (b) Two refinements per patch. (c) Four refinements per patch.

	Cache Aware Smoother	Cache Aware MG
Full Domain	1.3530	1.6335
One Patch	1.2522	1.7120
Two Patches	1.2550	1.8221
Four Patches	1.2415	1.6780

Table I. Itanium 1 speedups versus standard multigrid.

	Cache Aware Smoother	Cache Aware MG
Full Domain	1.2080	1.4057
One Patch	1.1697	1.5791
Two Patches	1.1519	1.6245
Four Patches	1.1352	1.5176

Table II. Itanium 2 speedups versus standard multigrid.

refinement above that (although only two are illustrated). Geometric multigrid is used as the solver on the AMR base grid. The full domain refinement case starts on an 8×32 base grid and has a total of 7 grid levels.

The tables show results for the AMR multilevel method employing the *cache-aware* smoother and the *combined smoother* (as described in §5.2) compared with a standard implementation of the smoother. We call the combined smoother case *cache aware multigrid*.

The base grid discretizes the rectangle $[0,1] \times [0,4]$. The right hand side of the Poisson equation is chosen so that the solution is $u(x,y) = \sin(\pi x)\sin(\pi y/4)xe^{x^2+(y/4)^2}$ and the coefficient is $a(x,y) = 1 + \sin(\pi x)\sin(\pi y/4)xe^{x^2+(y/4)^2}$. The initial guess on the base grid is $u = 0$, and the convergence criteria is $||r_c|| < 10^{-6}||\rho_c||$.

Table I shows the results of the set of experiments run on an Itanium 1. Table II shows the results of the set of experiments run on a Itanium 2. Table III shows the results of the set of experiments run on a Pentium III. Table IV shows the results of the set of experiments run on

	Cache Aware Smoother	Cache Aware MG
Full Domain	1.4339	1.6402
One Patch	1.3500	1.8728
Two Patches	1.3161	1.8257
Four Patches	1.1478	2.1175

Table III. Pentium III speedups versus standard multigrid.

	Cache Aware Smoother	Cache Aware MG
Full Domain	1.1434	1.3000
One Patch	1.0675	1.3379
Two Patches	1.0713	1.4019
Four Patches	1.0548	1.2758

Table IV. Pentium IV speedups versus standard multigrid.

Machine	Speedup
Pentium III	2.3431
Pentium IV	1.2298
Itanium 1	1.8721
Itanium 2	1.5209

Table V. Speedups for the cache aware smoother alone.

a Pentium IV. These experiments show the speedups associated with the cache optimizations. The timings measure only the solution procedure. They do not include initialization of the hierarchy, coefficient matrix and right hand side. The total speedups are up to about a factor of two. Table V shows the speedups associated with performing smoothing alone, outside of the context of a multilevel method.

The speedups (not shown) for doing post-smoothing only versus doing pre-smoothing *and* post-smoothing are around 20% with the cache aware smoother. Note that both cases do the same total number of smoothing iterations per level. In particular, the pre-/post-smoothing case uses 2 smoothing iterations on each side of the V. The post-smoothing only case uses 4 smoothing iterations on only one side of the V.

6.1. Conclusions

We experimented with cache aware smoothers and integration of the residual computation with the cache aware smoother. Both of these give good speedups in many cases. The integration of the residual computation is especially useful for getting good speedups on AMR hierarchies.

Modifying the algorithm to do post-smoothing only is key to realizing good performance in the AMR context. It takes better advantage of the cache aware smoother since the smoothing iterations on each level are all contiguous.

Future plans for this research include experimenting with three dimensional problems and parallel algorithms. Progress is already being made toward both of those topics. In addition, we are going to experiment with a variety of other cache optimization techniques (beyond mere

row based blocking). The 3D case, in particular, needs more sophisticated approaches, such as those discussed in [7].

REFERENCES

1. S. Agmon, *Lectures on Elliptic Boundary Value Problems*, Van Nostrand Reinhold, New York, 1965.
2. M. J. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, J. Comput. Phys., 82 (1989), pp. 64–84.
3. M. J. Berger and J. Oliger, *An adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 53 (1984), pp. 484–512.
4. W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, SIAM Books, Philadelphia, 2000. Second edition.
5. M. S. Day and J. B. Bell, *Numerical Simulation of Laminar Reacting Flows with Complex Chemistry*, Combust. Theory Modelling, 4 (2000), pp. 535–556.
6. C. C. Douglas, G. Haase, and U. Langer, *A Tutorial on Elliptic PDE's and Their Parallelization*, vol. 16 of Software, Environment, and Tools series, SIAM Books, Philadelphia, 2003.
7. C. C. Douglas, J. Hu, J. Ray, D. T. Thorne, and R. S. Tuminaro, *Fast, adaptively refined computational elements in 3D*, in Computational Science – ICCS 2002,, vol. 3, Berlin, 2000, Springer–Verlag, pp. 774–783. Check publisher.
8. S. A. Dudek and P. Colella, *Steady-state solution-adaptive euler computations on structured grids*, Tech. Report LBNL-41343, Lawrence Berkeley National Laboratory, 1998. Also presented at the 36th AIAA Aerospace Sciences Meeting and Exhibit, January 1998, Reno, NV, paper no. AIAA-98-0543.
9. D. Martin and K. Cartwright, *Solving Poisson's equation using adaptive mesh refinement.* LBL research report, see http://seesar.lbl.gov/anag/staff/martin/tar/AMR.ps, 1996.
10. S. F. McCormick, *The fast adaptive composite (FAC) method for elliptic equations*, Math. Comp., 46 (1986), pp. 439–456.
11. K. W. Morton and D. F. Mayers, *Numerical Solution of Partial Differential Equations*, Cambridge University Press, Cambridge, 1994.
12. J. J. Quirk and S. Karni, *On the dynamics of a shock-bubble interaction*, J. Fluid Mech., 318 (1996), pp. 129–163.
13. J. Ray, C. Kennedy, S. Lefantzi, and H. N. Najm, *High-order spatial discretizations and extended stability methods for reacting flows on structured adaptively refined meshes*, in Third Joint Meeting of the U.S. Sections of The Combustion Institute, Chicago, Illinois, March 2003. Distributed as on a CD.
14. U. Rüde, *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, vol. 13 of Frontiers in Applied Mathematics, SIAM, Philadelphia, 1993.
15. R. Samtaney, S. Jardin, P. Colella, and T. Ligocki, *High-resolution adaptive mesh simulations of magnetic reconnection*, in Bulletin of the American Physical Society, Division of Plasma Physics, American Physical Society, 2002. DPP meeting, Orlando, FL, Nov 11-15, 2002.
16. D. T. Thorne, *Multigrid with Cache Optimizations on Adaptive Mesh Refinement Hierarchies*, PhD thesis, University of Kentucky, 2003. In preparation.
17. R. S. Varga, *Matrix Iterative Analysis*, Prentice–Hall, Englewood Cliffs, NJ, 1962.